SpectreRewind: Leaking Secrets to Past Instructions

Jacob Fustos, Michael Bechtel, Heechul Yun

University of Kansas, USA

KU-CSL

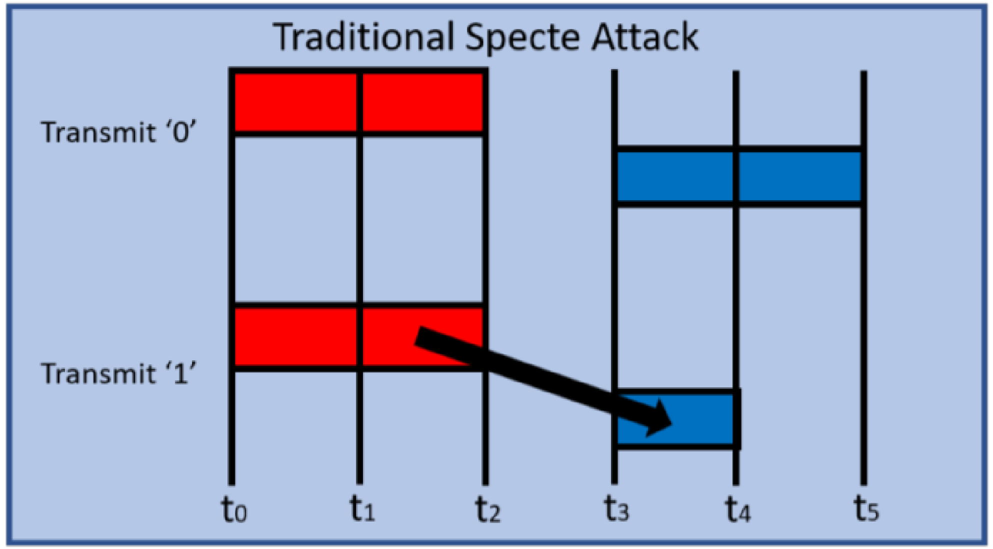
Computer Systems Lab

Speculative Execution Attacks

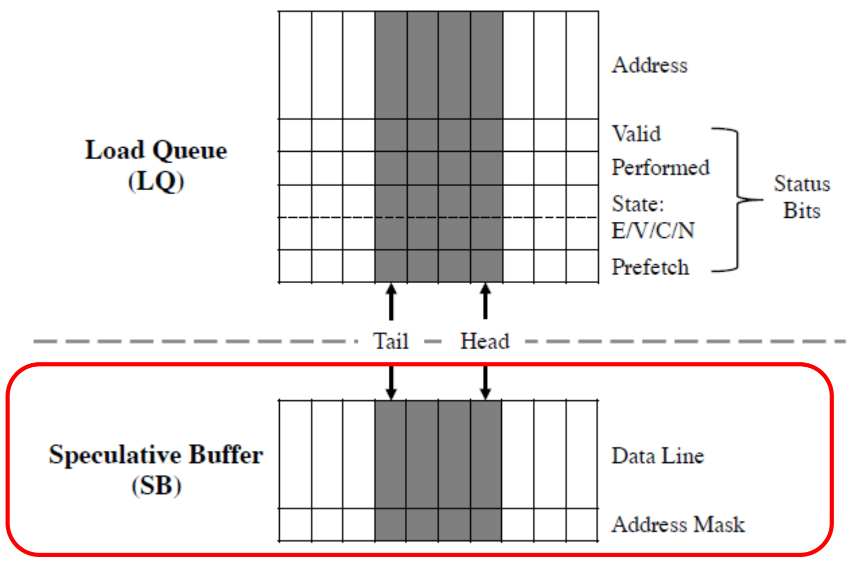
- Attacks exploiting microarchitectural side-effects left by speculative (transient) instructions
- Many variants: Spectre, Meltdown, Foreshadow, MDS, LVI, …
- Secrets are transferred over microarchitectural covert channels
- Most known attacks use cache covert channels

Cache-based Covert Channels

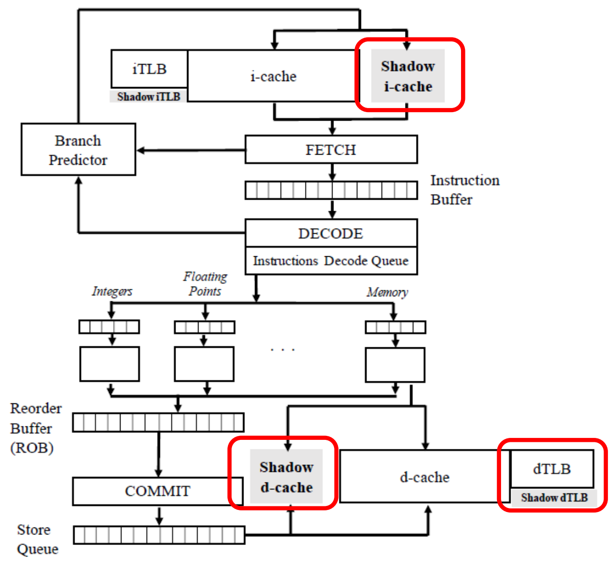
- Exploit that speculatively executed memory instructions change cache
- Encode secret by executing secret dependent memory accesses
- Use cache hit/miss timing differences to recover the secret
  - Cache miss → 0 (was not accessed)
  - Cache hit → 1 (was accessed)
- Secret is recovered ***after*** transient executions are squashed



Cache Covert Channel Mitigation Solutions



InvisiSpec [Yan et al., MICRO'18]

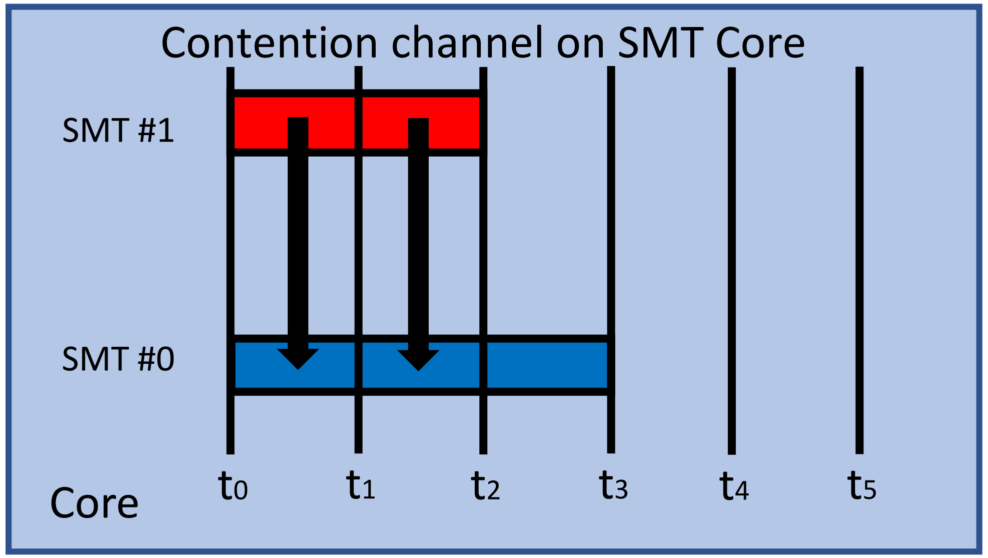


SafeSpec [Khasawneh et al., DAC'19]

- Prevent speculative execution from modifying the cache
  - Buffer speculative results in additional "*shadow*" hardware structures
- Undo cache changes. E.g., CleanupSpec [Saileshwar et al., MICRO'19]

Contention-based Covert Channels on SMT

- Exploit that contention on shared functional units/ports between Simultaneous multithreading (SMT) hardware threads
- Secret is transmitted ***during*** the speculative execution
  - Bypass stateful covert channel defenses such as InvisiSpec
- Can be mitigated by
  - Disabling SMT
  - Preventing co-scheduling of different domains



e.g,. SmotherSpectre [Bhattacharyya+, CCS 19]

Outline

- Background
- **SpectreRewind**
- Evaluation
- Conclusion

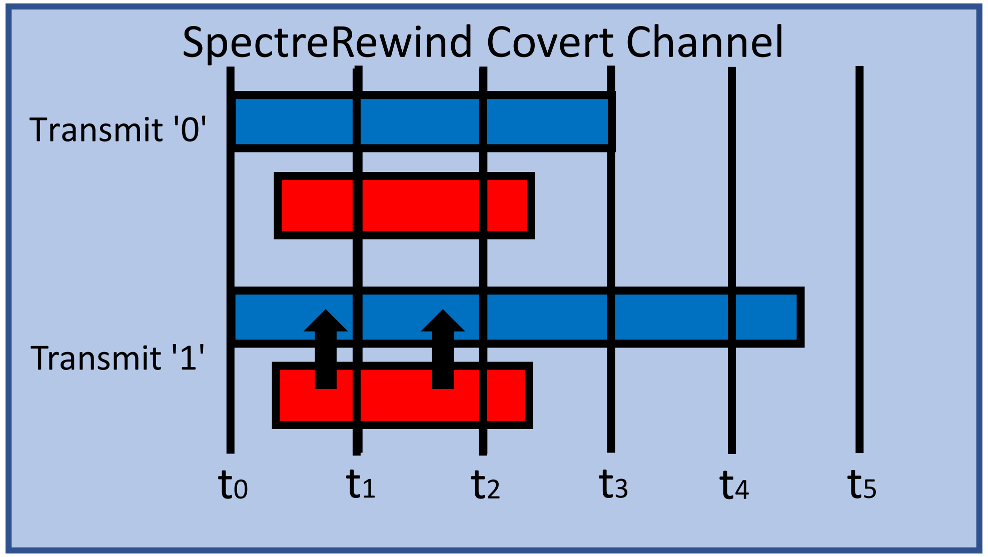
SpectreRewind

- A novel approach for creating contention-based covert channels
- From a single hardware thread (doesn't require SMT)
- Transmit secret to *past instructions*
- Bypasses existing cache covert channel defenses

- Key insight:
  - Contention-based covert channels w/o SMT are possible
  - Contention between speculative and past non-speculative instructions can create covert channels

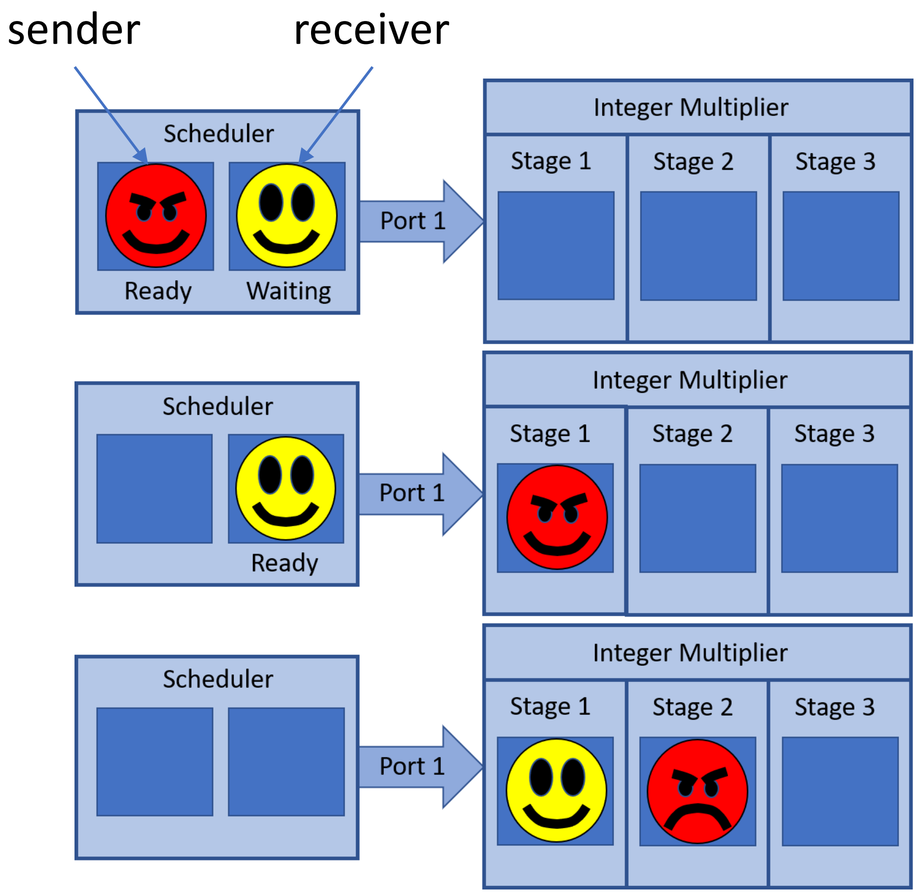
Our Approach

- Receiver: instructions executed *before* speculative execution
- Sender: instructions conditionally (secret dependent) executed *during* speculative execution
- Both sender and receiver are in the same hardware thread
- Contention between the sender and the receiver creates a channel
  - 0 – fast
  - 1 – slow (due to contention)
- Target *non-pipelined functional units*



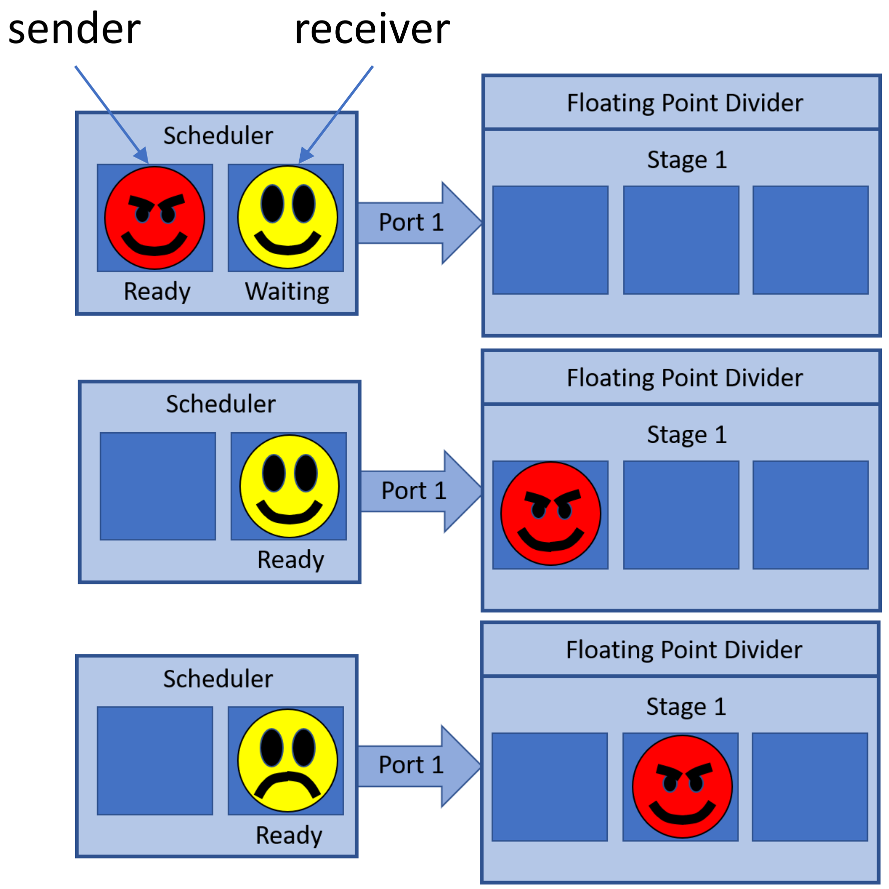
- Blue = Receiver, Red = Sender

Pipelined vs. Non-pipelined Function Units



(a) Fully pipelined

- No impact to receiver execution time

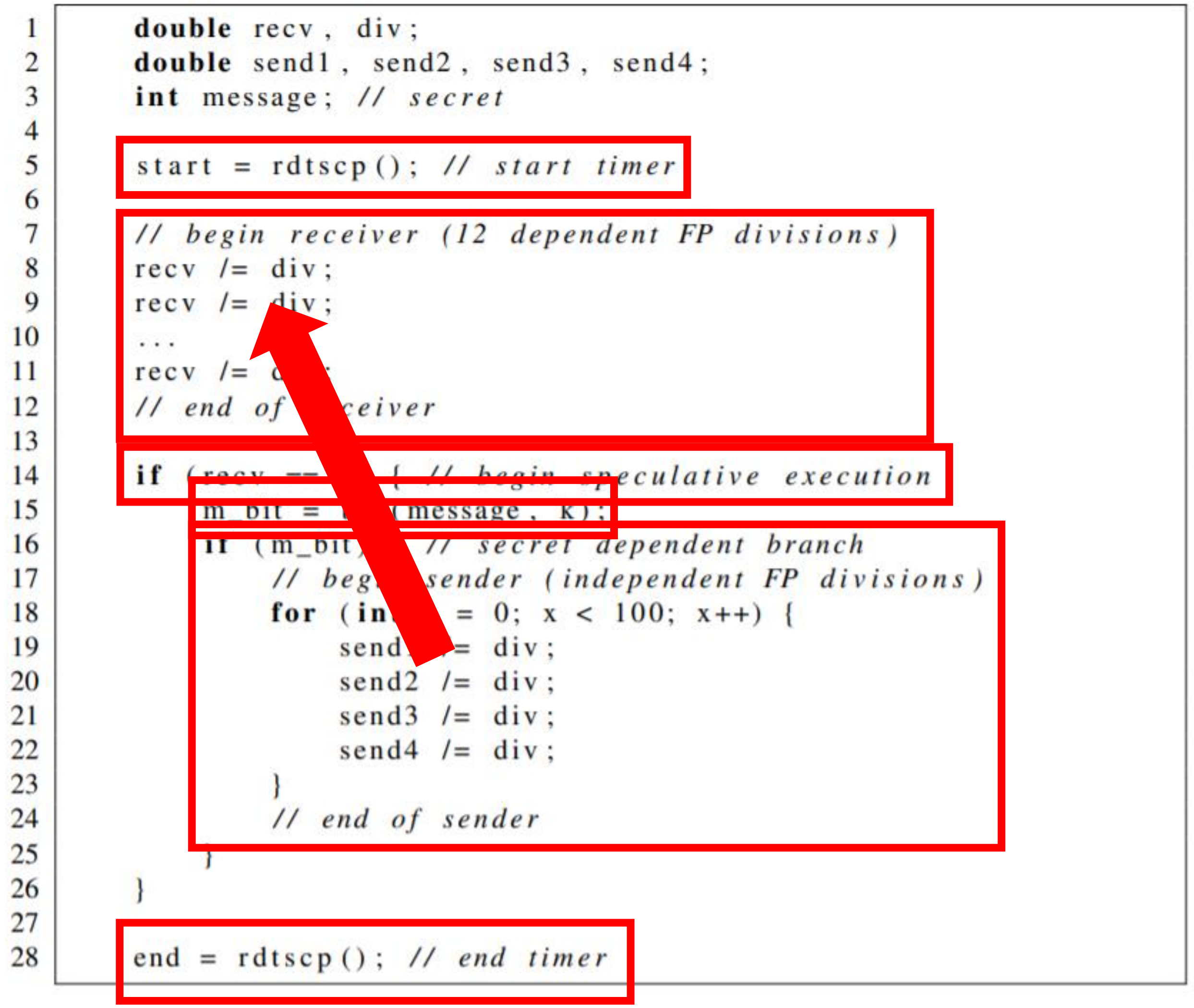


(b) Non-pipelined

- **Receiver execution time is prolonged**

Floating Point Division Covert Channel

- Start a timer
- Perform multiple divisions
- Cause a mis-speculation
- Calculate a bit to transmit
- If bit is ‘1’ do more division
- Cause contention with receiver
- Time entire attack



```
double recv, div;
double send1, send2, send3, send4;
int message; // secret

start = rdtscp(); // start timer

// begin receiver (12 dependent FP divisions)
recv /= div;
recv /= div;
...
recv /= d  ;
// end of   ceiver

if (recv ==   { // begin speculative execution
    m_bit =    (message, k);
    if (m_bit)   // secret dependent branch
        // beg   sender (independent FP divisions)
        for (in    = 0; x < 100; x++) {
            send    = div;
            send2 /= div;
            send3 /= div;
            send4 /= div;
        }
        // end of sender
    }
}

end = rdtscp(); // end timer
```

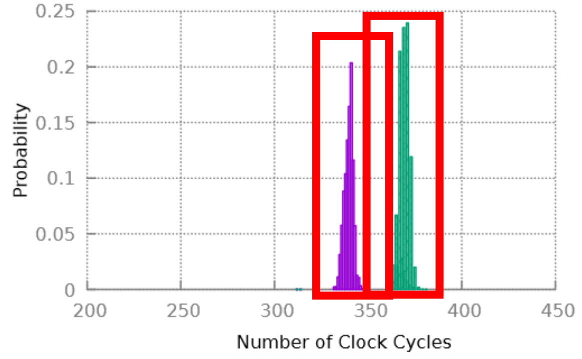
Outline

- Background
- SpectreRewind
- **Evaluation**
- Conclusion

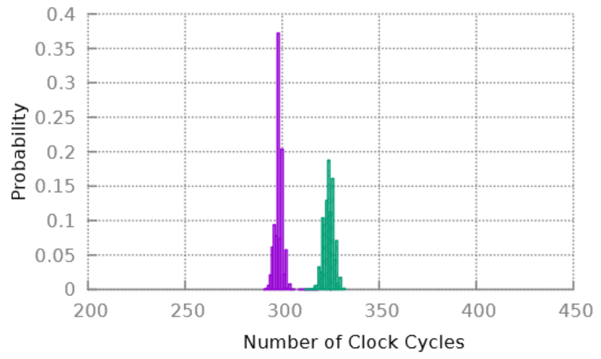
Evaluation Platforms and Methodology

- Used eight platforms from Intel, AMD, and ARM
- Native SpectreRewind PoC
  - Written in standard C language
- Timing measurement methods
  - RDTSC instructions on Intel and AMD (x86-64)
  - A counting thread on ARM (arm64)
- Methodology
  - Send 1 million known ‘0’ and ‘1’ bit values over the division channel and measure the timings

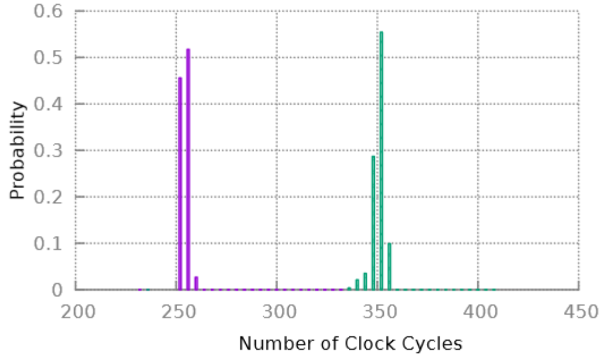
Channel Properties



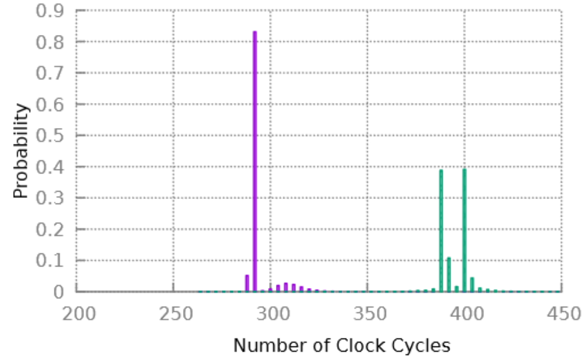
(a) Kabylake R (i5-8250U)



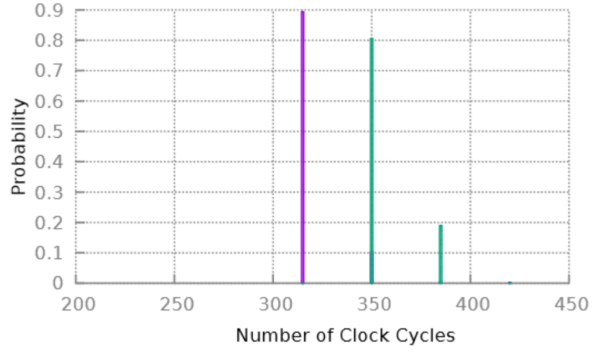
(b) Skylake (i5-6500)



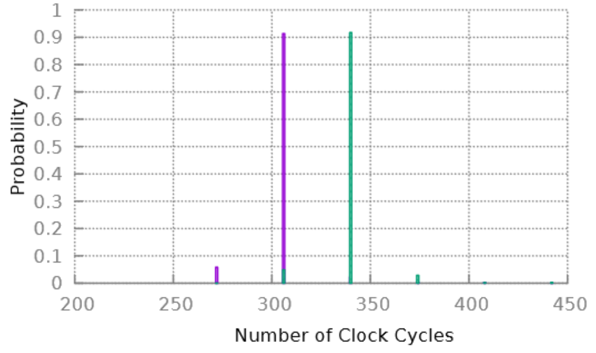
(c) Haswell (E5-2658v3)



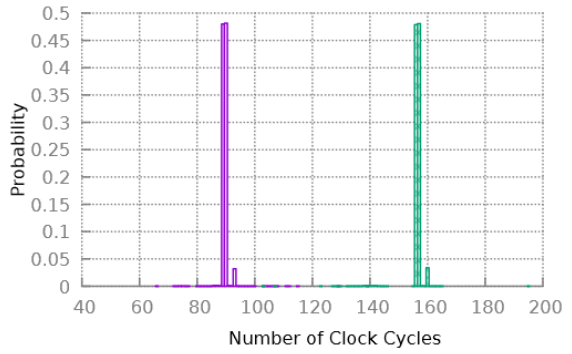
(d) Ivybridge (i5-3340M)



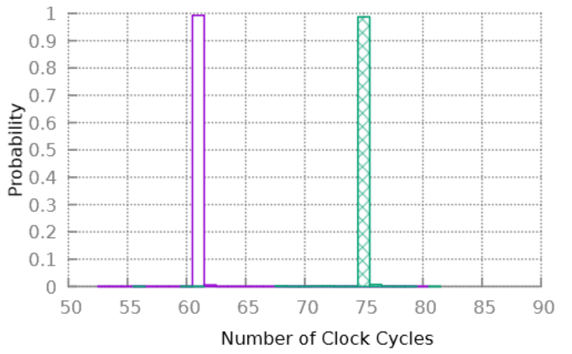
(e) Zen (Ryzen3 2200G)



(f) Zen+ (Ryzen5 2600)



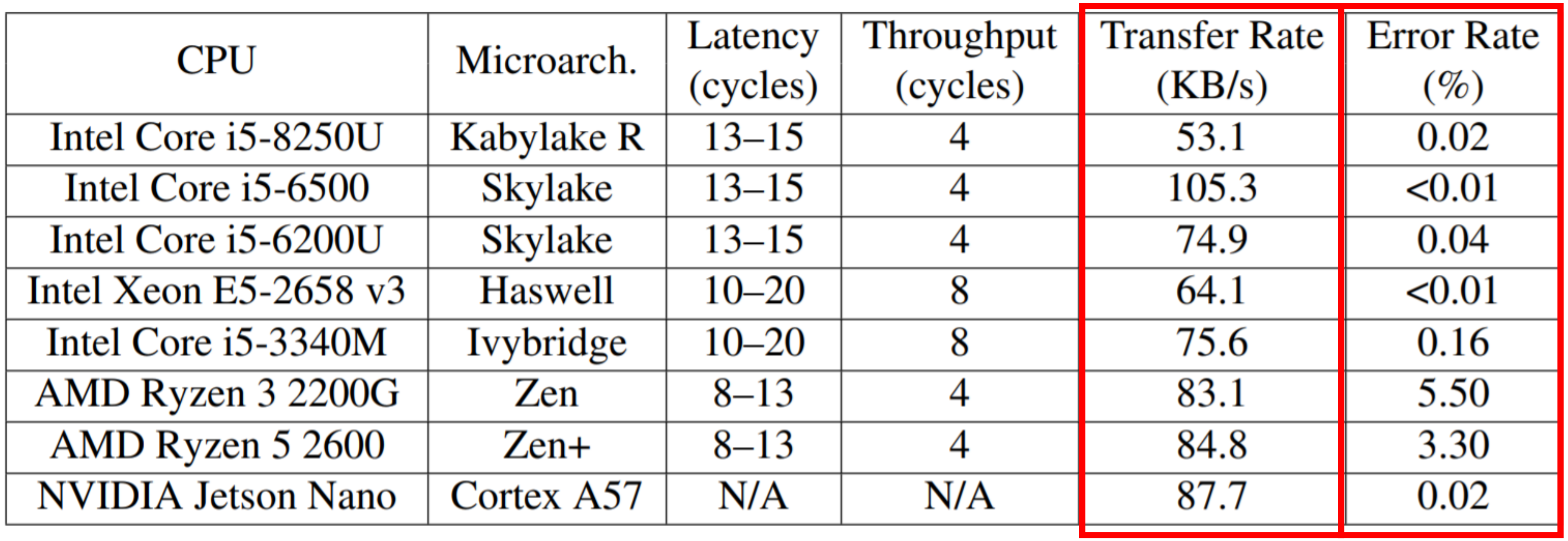
(g) Cortex-A57 (Jetson Nano)*



(h) Cortex-A72 (Raspberry Pi 4)*

- Clearly distinguishable patterns on all tested platforms

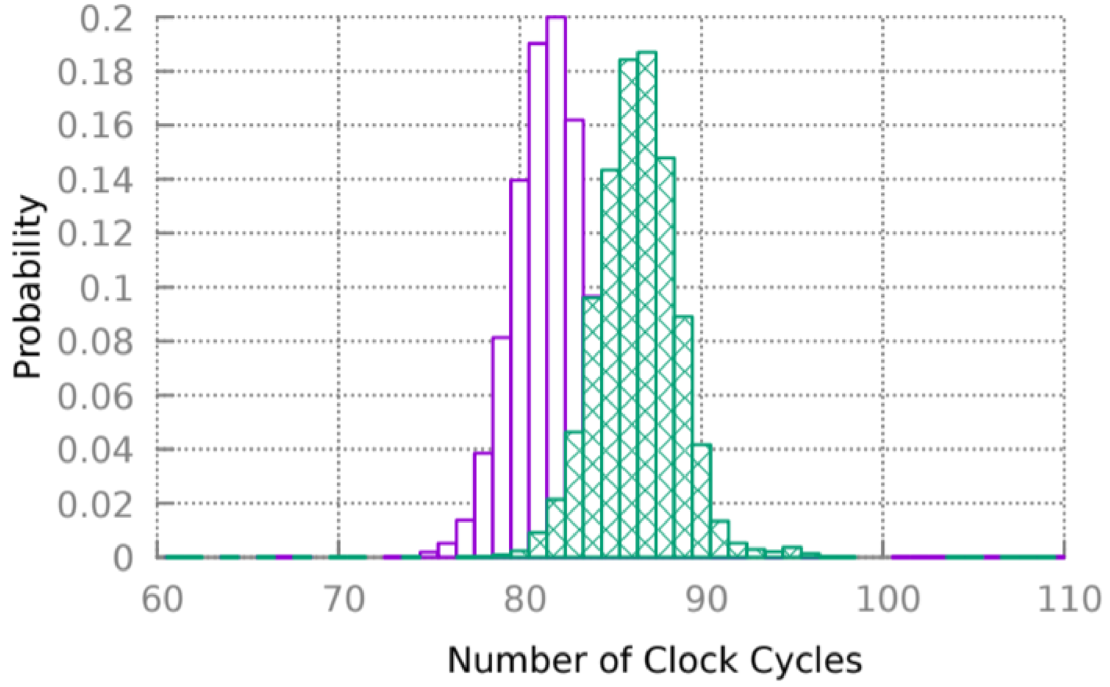
Performance Analysis



CPU	Microarch.	Latency (cycles)	Throughput (cycles)	Transfer Rate (KB/s)	Error Rate (%)
Intel Core i5-8250U	Kabylake R	13–15	4	53.1	0.02
Intel Core i5-6500	Skylake	13–15	4	105.3	<0.01
Intel Core i5-6200U	Skylake	13–15	4	74.9	0.04
Intel Xeon E5-2658 v3	Haswell	10–20	8	64.1	<0.01
Intel Core i5-3340M	Ivybridge	10–20	8	75.6	0.16
AMD Ryzen 3 2200G	Zen	8–13	4	83.1	5.50
AMD Ryzen 5 2600	Zen+	8–13	4	84.8	3.30
NVIDIA Jetson Nano	Cortex A57	N/A	N/A	87.7	0.02

- High transfer rates and low error rates

Google Chrome Sandbox



- Implemented a SpectreRewind PoC in JavaScript on Chrome
- Noisier but still distinguishable timing differences

Outline

- Background
- SpectreRewind
- Evaluation
- **Conclusion**

Discussion

- Benefits
    - Does not require SMT hardware (single thread)
    - Defeats all known hardware solutions for stateful (cache) covert channels
    - Alternative to cache-based covert channels like Flush+Reload
- Limitations
    - Limited to same address space attacks
    - Finding division-based gadgets may be difficult
    - Attacker controls both receiver and sender

Conclusion

- A novel approach for creating contention-based covert channels
  - Transmit secret to *past instructions*
  - Bypass existing cache covert channel defenses
- Exploits contention on *non-pipelined functional units*
  - Between speculative and past non-speculative instructions
  - From a single hardware thread
- Floating point division unit based covert channel
  - ~100KB/s transfer rate, <1% error
  - Works across CPUs from Intel, AMD, and ARM